Microsoft
.net developer
training
Create XML Web Services and Applications with Visual Studio .NET

Code Instrumentation and Middle Tier Technologies

Agenda: Day 2

- Module 09: Creating XML Web Services
- ▶ Module 10: Code Instrumentation and Middle Tier Technologies
- Module 11: Security in .NET
- Module 12: Lifecycle Tools for .NET
- Module 13: Configuring and Deploying .NET Applications
- Module 14: Application Interoperability
- Module 15: Migrating Applications to .NET
- Module 16: Building Applications for Mobile Devices

Part # X08-76394
©2002 Microsoft Corporation

Module 10 Agenda

What you will learn:

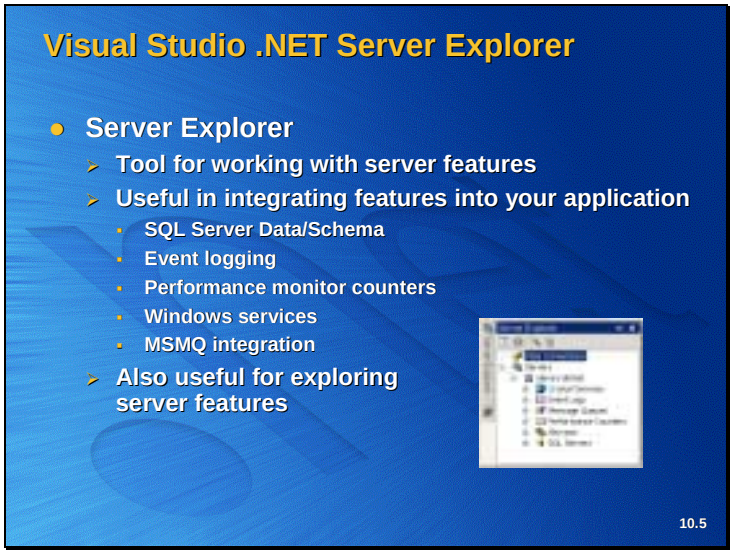
How to use the Server Explorer to integrate features into your applications

- MSMQ integration
- Windows services
- Event logging
- Performance monitor counters

Application Goals

The importance of code instrumentation (Tracing, Perf Counters, Event Log) grows rapidly as server applications become (became) mainstream. Server applications do not have a “view” (like client applications), so they are hard to test. The developer cannot just start the application and try it; there is more to server applications than a single user code path. Also lots of server application problems manifest themselves only in real life environment (under real customer load).

 ©2002 Microsoft Corporation

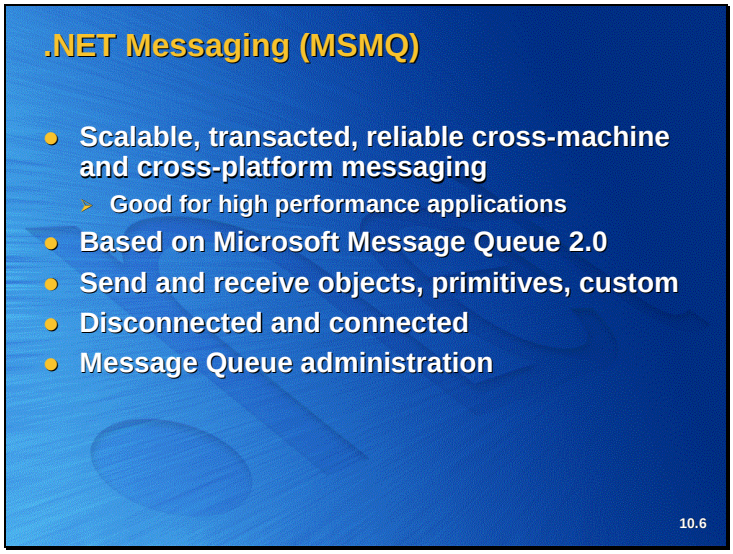


Visual Studio .NET Server Explorer

Server Explorer is a new server-development console for Visual Studio .NET. It is a shared tool window that helps developer's access and manipulates resources on any computer for which they have permission.

In Visual Basic, you would construct forms by dragging controls onto them, setting their properties, and then using them in code.

With .NET, you can now do the same with UI-less server components.

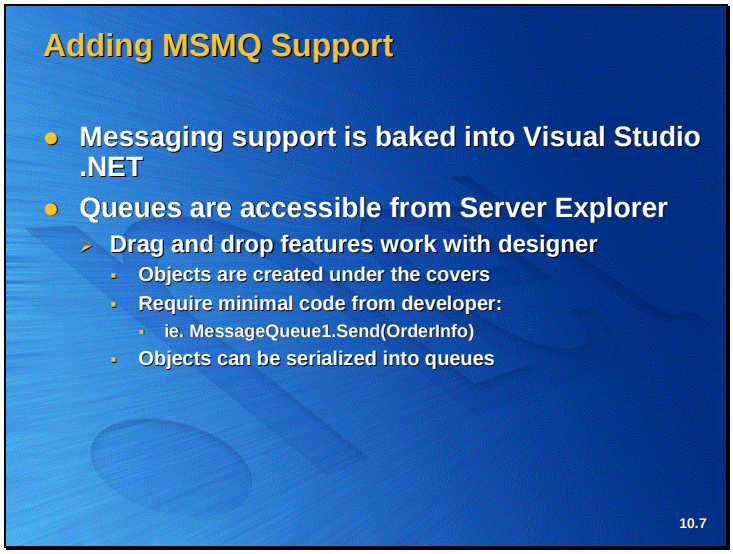

.NET Messaging (MSMQ)
Scalable, transacted, reliable cross-machine and cross-platform messaging
Good for high performance applications
Based on Microsoft Message Queue 2.0
Send and receive objects, primitives, custom
Disconnected and connected
Message Queue administration
10.6

.NET Messaging (MSMQ)

Message Queue is a wrapper around **MSMQ 2.0 (Win2K)** and **MSMQ 1.0 (WinNT).** Current plans for **MSMQ 3.0,** which will ship with Windows .NET Server, is for it to provide functionality parallel to that of **MSMQ ActiveX** control and also provide the benefits of the managed world. MSMQ 3.0 is already faster, and there remains time for improvements before the program's ship date.

Two lines of code are enough to send an instance of any COM+ 2.0 class to a message queue and another two lines of code to receive on the other side of the wire.

 ©2002 Microsoft Corporation


Adding MSMQ Support
Messaging support is baked into Visual Studio .NET
Queues are accessible from Server Explorer
Drag and drop features work with designer
Objects are created under the covers
Require minimal code from developer:
ie. MessageQueue1.Send(OrderInfo)
Objects can be serialized into queues
10.7

Adding MSMQ Support

MSMQ technology enables applications running at different times to communicate across heterogeneous networks and systems that temporarily may be offline. Applications send messages to queues and read messages from queues. MSMQ provides guaranteed message delivery, efficient routing, security and priority-based messaging. It can be used to implement solutions for both asynchronous and synchronous scenarios requiring high performance.

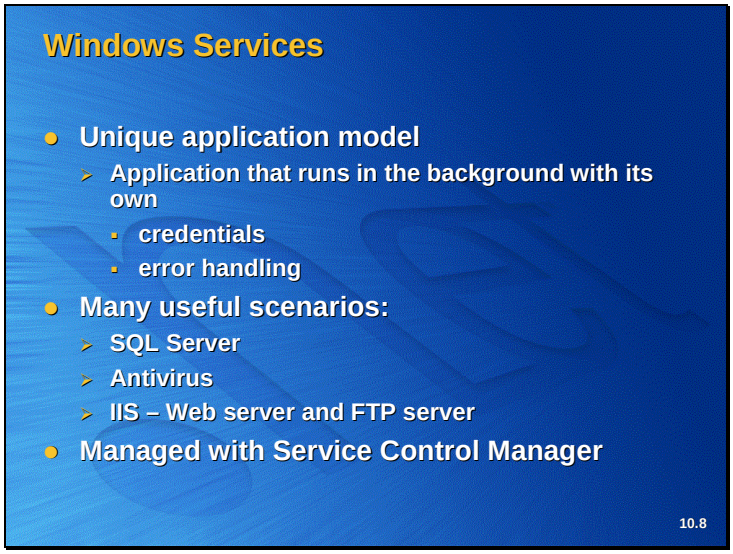
MessageQueue can work with **XmlMessageFormatter**, which allows messaging client and server to be decoupled and versioned independently.

MSMQ is the easiest message queuing product to use, deploy and administer on the Windows 2000 Platform because of its full COM support, dynamic directory service-based architecture and built-in centralized systems management.

Category - Framework services

Namespace - System.Messaging

Functionality - Offers comprehensive functionality such as reliable, resilient message delivery, cost-based message routing and full support for transactions. MSMQ also offers seamless interoperability with the IBM MQSeries products through the Microsoft MSMQ-MQSeries Bridge and with other message queuing products through products from Level 8 Systems, Inc.
With **Server Explorer**, developers can connect to servers and view their resources including message queues, performance counters, services, processes, event logs and database objects.

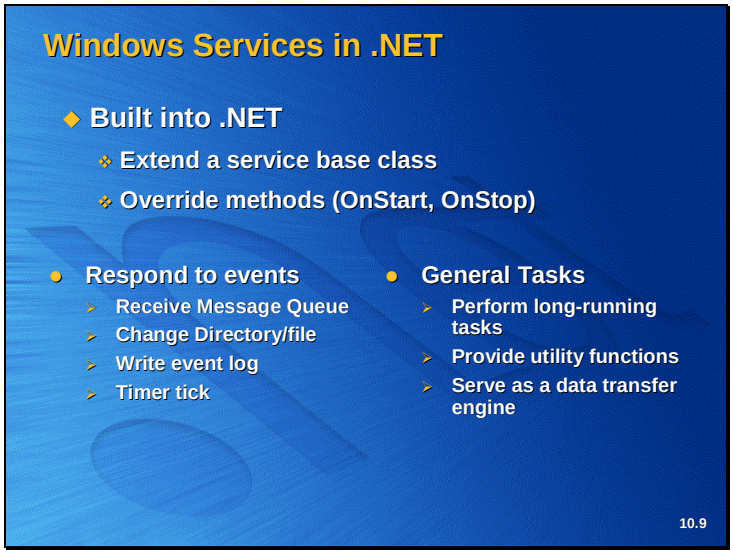


Windows Services

Windows services generally do most of their work in the background based on preconfigured instructions and schedules. They are configured and administered via a management console, for example, the Microsoft Management Consol, MMC, which allows you to administer IIS, SMTP, FTP, and Content Indexer (CI). You start and stop services using the Service Control Manager. You also administer Windows services from the Service Control Manager, for example assigning them IDs and passwords, setting options such as whether the service starts automatically when the system boots, or requires a manual startup.

It is possible to have more than one Windows service per process.

 ©2002 Microsoft Corporation



Windows Services in .NET

Windows services are useful for certain scenarios (primarily non-Web server applications). Earlier, having .NET implement a service was complex or required the use of a tool like SRVANY. Now you can derive from a service base class, override some methods and detect new files that were sent to a server through an FTP.

```
Inherits ServiceBase
Overrides Sub OnStart()
    Timer1.Enabled = True
End Sub
Overrides Sub OnStop()
    Timer1.Enabled = False
End Sub
Sub Timer1_Tick()
    'do work
End Sub
```

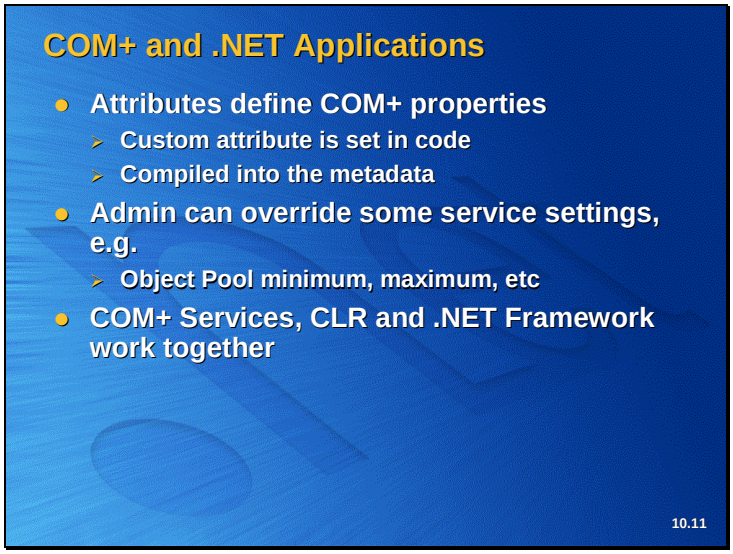
Windows Services

Demo

The purpose of this demo is to illustrate design goals:

- UI-less and unattended
- Long running and cannot block on start/stop/pause
- Can work with different credentials

 ©2002 Microsoft Corporation


COM+ and .NET Applications
Attributes define COM+ properties
Custom attribute is set in code
Compiled into the metadata
Admin can override some service settings, e.g.
Object Pool minimum, maximum, etc
COM+ Services, CLR and .NET Framework work together
10.11

COM+ Attribute Program Model

Instances of a .NET Framework class can participate in an automatic transaction, as long as you prepare the class to do so. Each resource accessed by a class instance, or object, enlists in the transaction. For example, if an object uses ADO.NET to post money on an account in a database, the resource manager for the database determines whether the object should execute in a transaction. If so, it automatically enlists the database in the transaction.

Use the following process to prepare a class to participate in an automatic transaction:

1. Apply the TransactionAttribute to your class.
2. Derive your class from the ServicedComponent Class.
3. Sign the assembly with a strong name.

   To sign the assembly using attributes create a key pair using the Sn.exe utility.

   sn -k **TestApp.snk**
4. Add the **AssemblyKeyFileAttribute** or **AssemblyKeyNameAttribute** assembly attribute specifying the name of the file containing the key pair to sign the assembly with a strong name.

   [Visual Basic]

   <assembly: AssemblyKeyFileAttribute("**TestApp.snk**")>

   [C#]

   [assembly: AssemblyKeyFileAttribute("**TestApp.snk**")]

   For details see Signing an Assembly with a Strong Name.
5. Register the assembly that contains your class with the COM+ catalog.

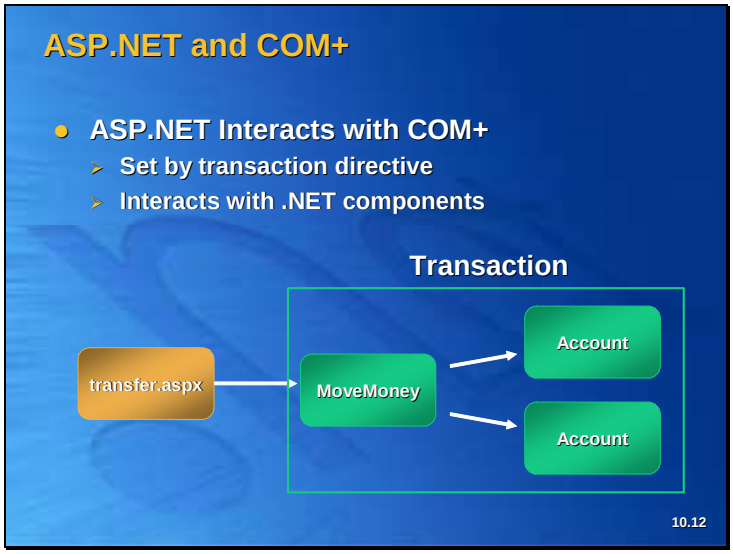
   If the client calling instances of your class is managed by the common language runtime, the registration is performed for you. However, if you anticipate that an unmanaged caller may create and call instances of your class, use the .NET Services Installation Tool (Regsvcs.exe) to perform the registration manually.

The following example shows how to apply the **TransactionAttribute** to a class derived from the **ServicedComponent** class.

[Visual Basic]
```
Public Class <Transaction(TransactionOption.Required)> Bar
   Inherits ServicedComponent
   '. . .
End Class
```
[C#]
```
[Transaction(TransactionOption.Required)]
public class Bar(): ServicedComponent
{
  //. . .
}
```

For more details on ServicedComponents see the .Net Framework SDK Documentation.

 ©2002 Microsoft Corporation



ASP.NET and COM+

XCOPY deployment

Dynamic registration

Managed code ASP+ Web Form (C#)

Managed code components (C#)

- Transactions
- AutoComplete

ASP.NET supports automatic transactions on systems running Microsoft Windows 2000. By inserting a transaction directive in your ASP.NET page, you can instruct the page to participate in an existing transaction, begin a new transaction, or never participate in a transaction. The following table lists and describes the transaction directives available in ASP.NET:

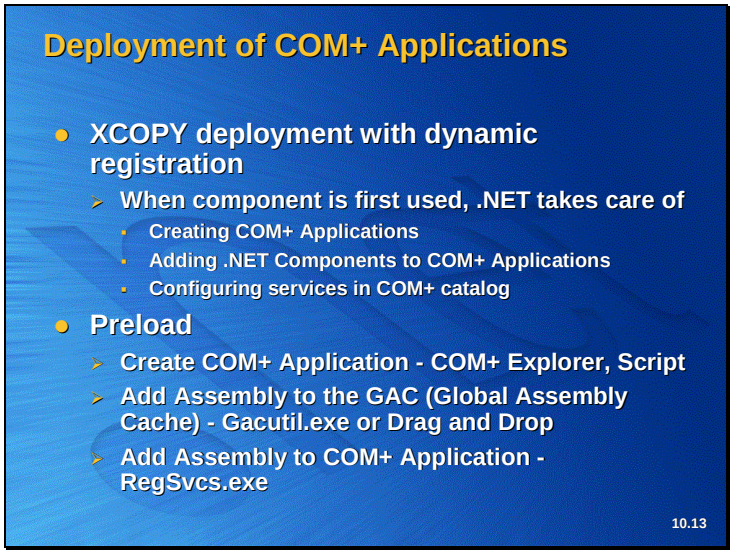
Directive	Description
Disabled	Indicates that the transaction context will be ignored by ASP.NET. This is the default transaction state.
NotSupported	Indicates that the page does not run within the scope of transactions. When a request is processed, its object context is created without a transaction, regardless of whether there is a transaction active.
Supported	Indicates that the page runs in the context of an existing transaction. If no transaction exists, the page runs without a transaction.
Required	The page runs in the context of an existing transaction. If no transaction exists, the page starts one.
RequiresNew	Indicates that the page requires a transaction and a new transaction is started for each request.

You can indicate the level of transaction support on a page by placing the directive in your code. For example, you can ensure that the page activities always execute in the scope of a transaction by inserting the following directive:

```
<%@ Page Transaction="Required" %>
```

If you omit the transaction directive, transactions are disabled for the page.

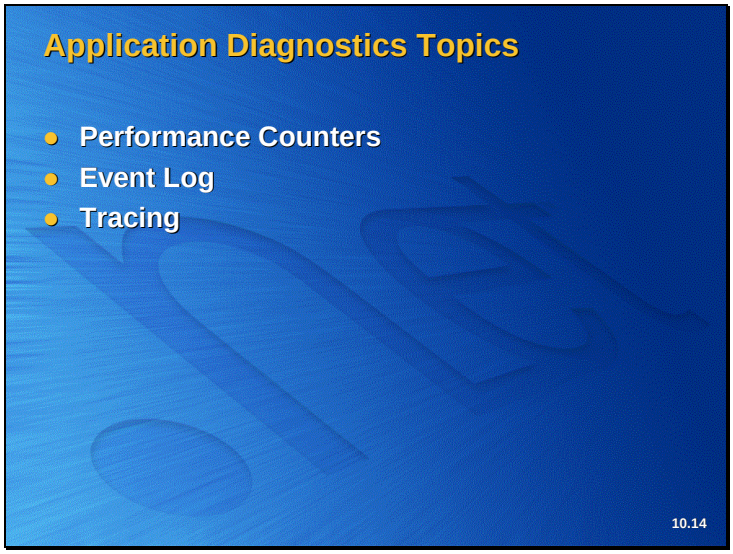
 ©2002 Microsoft Corporation


Deployment of COM+ Applications
XCOPY deployment with dynamic registration
When component is first used, .NET takes care of
Creating COM+ Applications
Adding .NET Components to COM+ Applications
Configuring services in COM+ catalog
Preload
Create COM+ Application - COM+ Explorer, Script
Add Assembly to the GAC (Global Assembly Cache) - Gacutil.exe or Drag and Drop
Add Assembly to COM+ Application - RegSvcs.exe
10.13

Deployment of COM+ Applications

See Appendix A: Common Utilities, for more information.

If a class in .NET uses COM+ services, then it is required to be registered in a COM+ application in the COM+ catalog as a regular COM+ component. When an instance of the class is created within .NET, this registration is done automatically based on the attributes in the assembly. However, if a new instance of the class is required from an unmanaged environment, then the assembly must be registered manually. This can be achieved using the tool, regsvcs.



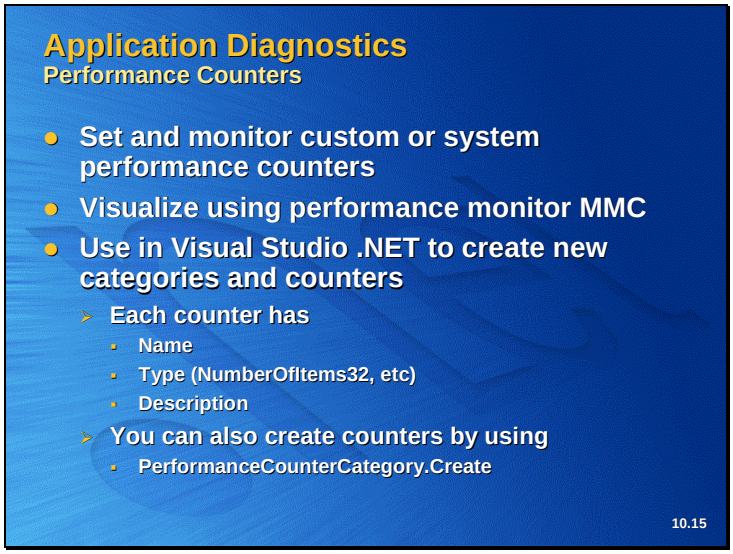
Application Diagnostics

Monitoring your application is important, especially in production.
The Windows 2000 and Windows NT System Monitor is an easy tool with which to profile your .NET application. The System Monitor graphically represents data found in the .NET performance counters that are installed with the .NET runtime and SDK. These counters can be used to monitor everything from memory management to just-in-time (JIT) compiler performance. They provide solid numbers that tell you exactly how fast your application is running without severely degrading your application's performance.

The easiest way to open the System Monitor is to type **"perfmon"** from the command line, which displays the Microsoft Management Console. Select the **Add** icon in the toolbar and select the performance counter you want to view.

Tracing is a flexible way to instrument your application as it helps debugging in situations where you cannot attach a debugger. Tracing is efficient at run time because you can control message levels.

 ©2002 Microsoft Corporation



Application Diagnostics

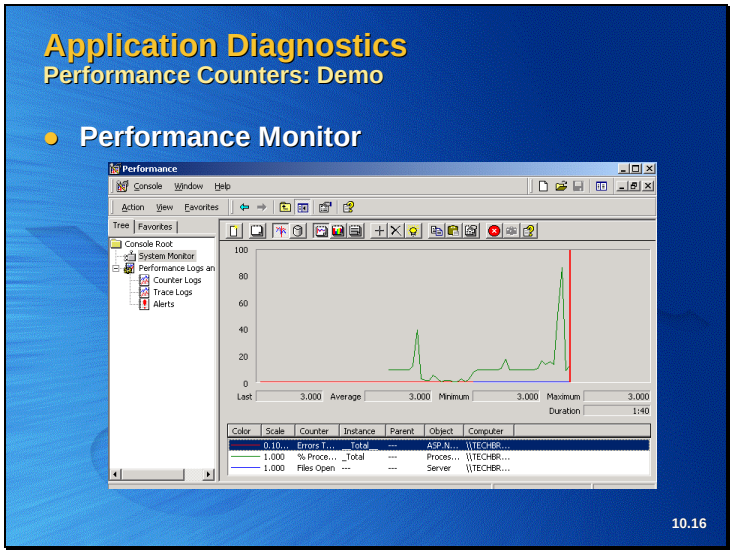
Performance Monitor

The .NET framework provides an API that you can use to access performance information directly from the built-in performance monitor or to create custom performance counters. The following performance monitoring classes are provided in the .NET framework:

- System.Diagnostics.PerformanceCounter Class
- System.Diagnostics.PerformanceCounterCategory Class
- System.Diagnostics.PerformanceCounterInstaller Class
- System.Diagnostics.PerformanceCounterManager Class
- System.Diagnostics.PerformanceCounterType Enumeration

PerformanceCounter Class

- Supports updating and reading counters
- Instantiates and Sets
  - CategoryName
  - CounterName
  - MachineName



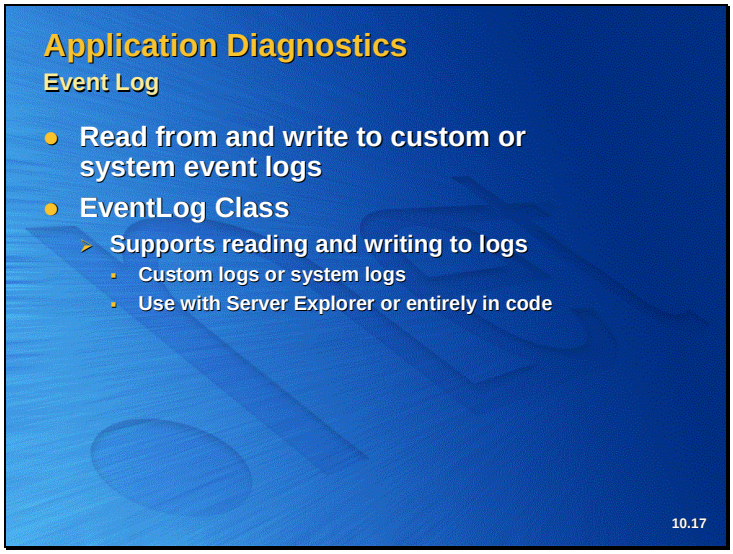
Application Diagnostics

Performance Counters: Demo

The purpose of this demo is to illustrate the Performance Monitor Counter:

- Add Performance Monitor Counter
- Discuss adding a custom counter

 ©2002 Microsoft Corporation


Application Diagnostics
Event Log
Read from and write to custom or system event logs
EventLog Class
Supports reading and writing to logs
Custom logs or system logs
Use with Server Explorer or entirely in code
10.17

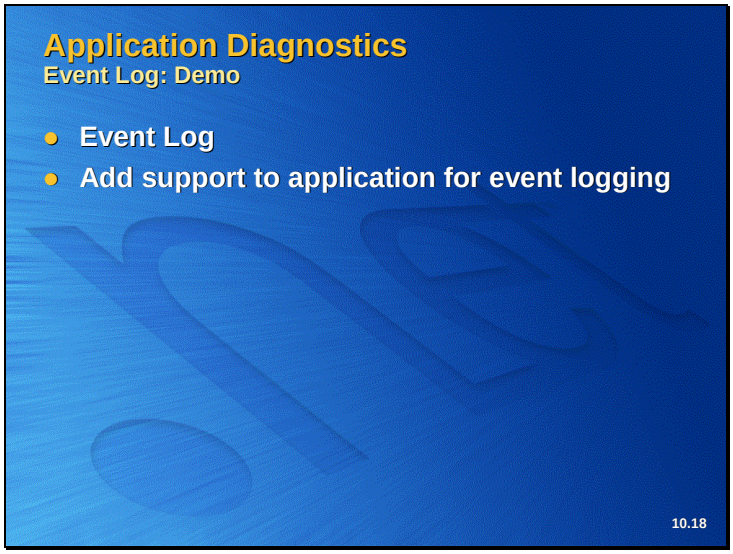
Application Diagnostics

Event Log

Setting log properties

Instantiates and Sets:

- Set Log = Custom or system log name
- Set MachineName = server
- Set Source = Application Name
- Call WriteEntry to write

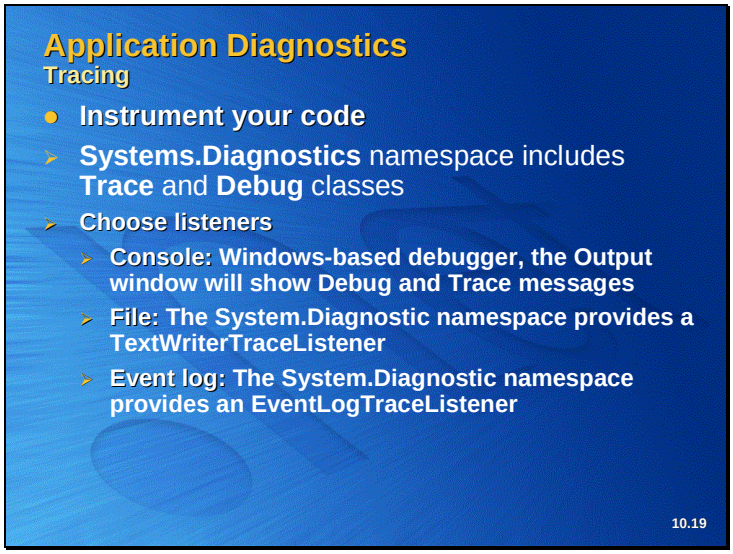


Application Diagnostics

Event Log: Demo

The purpose of this demo is to show how to add event log diagnostics to the DBLayer and Business layer.

 ©2002 Microsoft Corporation


Application Diagnostics
Tracing
Instrument your code
Systems.Diagnostics namespace includes Trace and Debug classes
Choose listeners
Console: Windows-based debugger, the Output window will show Debug and Trace messages
File: The System.Diagnostic namespace provides a TextWriterTraceListener
Event log: The System.Diagnostic namespace provides an EventLogTraceListener
10.19

Application Diagnostics

Tracing

Fortunately, you do not have to step through an application line-by-line to figure out what is happening. The **Systems.Diagnostics** namespace includes **Trace** and **Debug** classes. These two classes (which are essentially identical) include a number of static methods that can be used to cause your code to gather information about code-execution paths, code coverage, and even performance profiling. Both classes also provide an **Assert** method that checks for a condition and displays a message if the condition is **false**.

To use the **Trace** and **Debug** classes, perform the following three steps.

1. Define the **TRACE** or **DEBUG** symbol.
2. Create a new instance of the corresponding **Listener** object.
3. Add calls to **Trace** or **Debug** to your code.

- **Define —** To enable **Debug** or **Trace**, you must first define the corresponding symbol. This can be done at compile time by using the **/d(efine)** conditional compilation switch for C# (and the Managed Extensions for C++) or for Visual Basic.NET, as shown here:

```
csc /target:library /debug+ /d:TRACE math.cs vbc /target:library
/debug+ /d:TRACE=True math.vb
```

  The symbol can also be defined in the source code itself, as shown here:

```
#define TRACE // for C# or the Managed Extensions for C++ #Const
TRACE=1 ' for Visual Basic.NET
```

- **Create —** Both **Debug** and **Trace** outputs are captured by a **TraceListener**. By default, output is automatically sent to the **DefaultTraceListener**, which sends output to the Windows system debugger by using the **OutputDebugString** Windows API and to the debugger by using the **Debugger.Log** method. Messages sent using **Debugger.Log** has no effect if there is no debugger attached.

If you are operating inside the Windows-based debugger, the **Output** window will show **Debug** and **Trace** messages. However, if you are running outside the debugger—for instance, on a tester's computer—or if you want to save debugging information to a text-based log file, you must create a new **Listener** object and add it to the **Listeners** collection. The **System.Diagnostic** namespace provides an **EventLogTraceListener** and a **TextWriterTraceListener**. Creating a text file to handle debugging messages requires only the following single line of code.

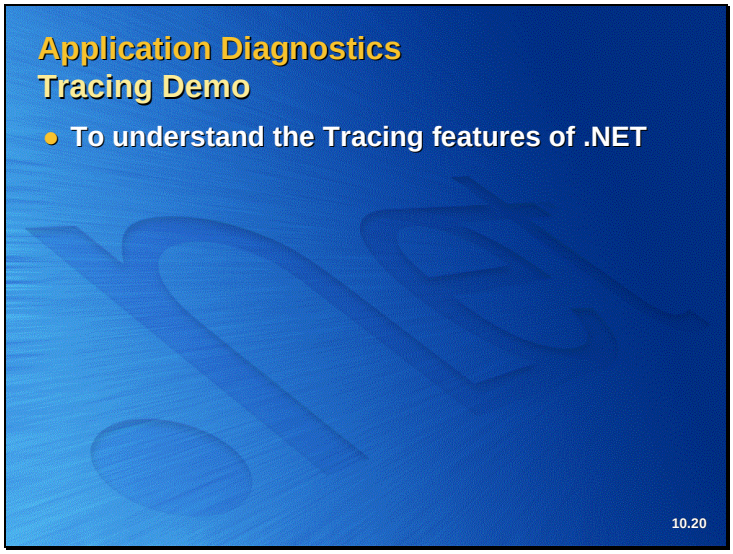
```
Trace.Listeners.Add(new
TextWriterTraceListener(File.Create("output.txt")));
```

- **Add—**Lastly, you can add calls to the various **Trace.Write** methods to generate debugging information at run time, as shown here:

```
Trace.WriteLine("");
Trace.WriteLine("Starting tracing...");
```

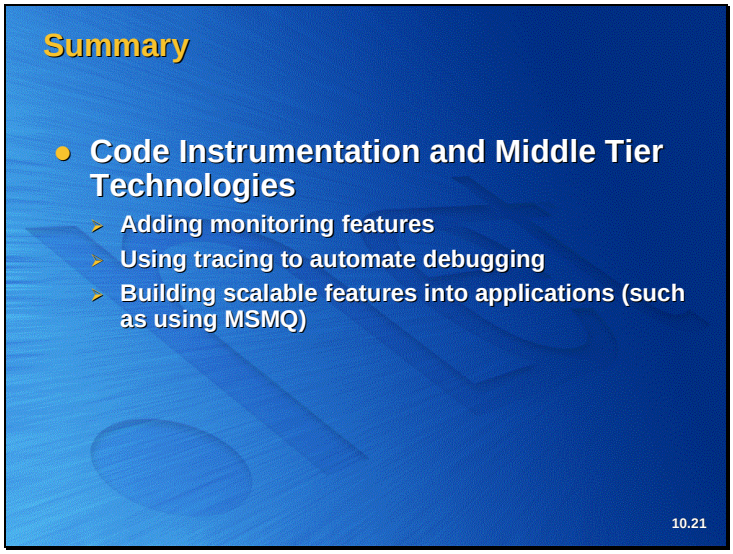
Since the first few bytes of the output file contain information about the text encoding of the file and are not readable, you might want to start logging on a new line, by first executing an empty **WriteLine** method.

 ©2002 Microsoft Corporation



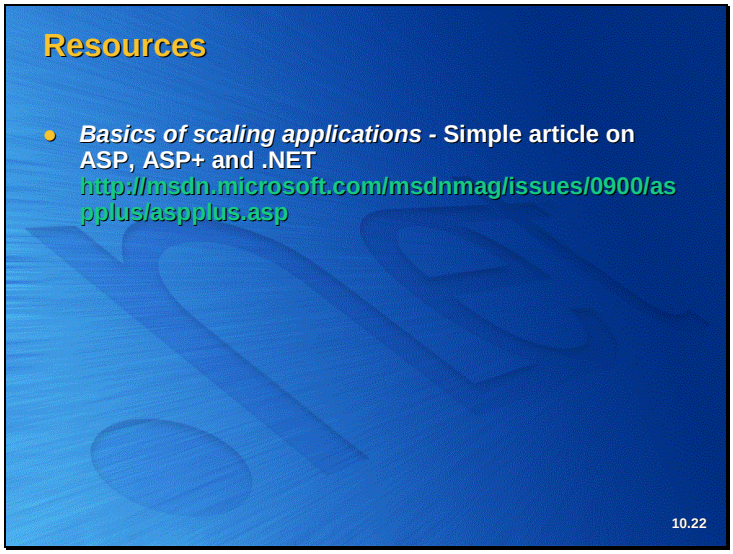
Tracing Demo

Objective: To understand the Tracing features of .NET


Summary
Code Instrumentation and Middle Tier Technologies
Adding monitoring features
Using tracing to automate debugging
Building scalable features into applications (such as using MSMQ)
10.21

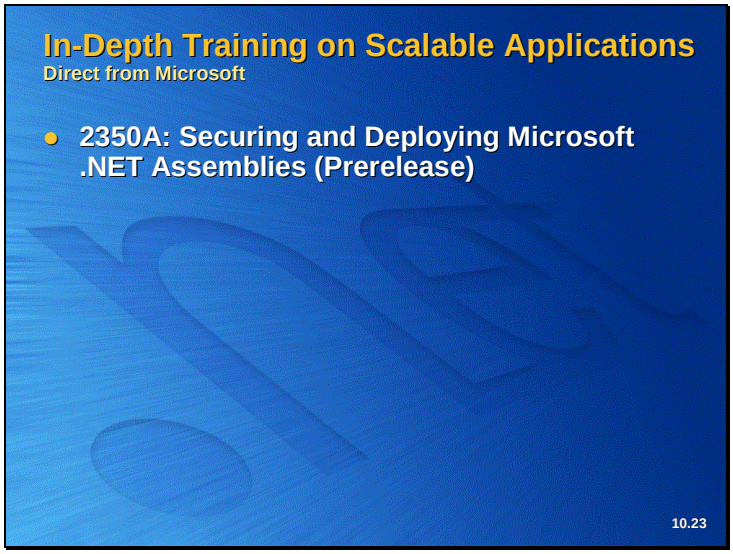
Summary

 ©2002 Microsoft Corporation



Resources

Basics of scaling applications - Simple article on ASP, ASP+ and .NET
http://msdn.microsoft.com/msdnmag/issues/0900/aspplus/aspplus.asp


In-Depth Training on Scalable Applications
Direct from Microsoft
2350A: Securing and Deploying Microsoft .NET Assemblies (Prerelease)
10.23

In-Depth Training on Scalable Applications

Direct from Microsoft

 ©2002 Microsoft Corporation

Microsoft
10.24